REMARKS

The present application was filed on July 3, 2001 with claims 1-25. Claims 1, 12, and 23-25 are independent. In the outstanding final Office Action, the Examiner: (i) rejected claims 1-10, 12-21, 23-25 under 35 U.S.C. §102(e) as being anticipated by U.S. Patent Application Publication No. US 2002/0194026 to Klein et al. (hereinafter "Klein").

Applicants acknowledge the indication of allowable subject matter in claims 11 and 22.

With regard to the rejection of claims 1-10, 12-21 and 23-25 under 35 U.S.C. §102(e) as being anticipated by Klein, Applicants submit herewith a declaration under 37 C.F.R. §1.131. The declaration is signed by the inventors named on the present application. The declaration and the exhibit attached thereto evidence the conception of an invention falling within independent claims 1, 12, 23-25 and at least one or more dependent claims, at least as early as March 14, 2001, and thus prior to the June 13, 2001 effective date of Klein. The declaration further evidences due diligence from March 14, 2001 until the filing date of July 3, 2001.

In view of the above, Applicants believe that claims 1-25 are in condition for allowance, and respectfully request withdrawal of the §102(e) rejection.

Respectfully submitted,

Date: April 29, 2004

Robert W. Griffith
Attorney for Applicant(s)
Reg. No. 48,956
Ryan, Mason & Lewis, LLP
90 Forest Avenue
Locust Valley, NY 11560
(516) 759-4547

YOR920010253US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s): G.G. Zweig et al.
Case: YOR920010253US1
Serial No.: 09/898,289
Filing Date: July 3, 2001
Group: 2182
Examiner: Justin R. Knapp

Title: Information Extraction from Documents with Regular Expression Matching

---

DECLARATION OF PRIOR INVENTION UNDER 37 C.F.R. §1.131

We, the undersigned, hereby declare and state as follows:

1. We are the named inventors on the above-referenced U.S. patent application.

2. The invention that is the subject of the present application was conceived at least as early as March 14, 2001. On or about this date, an IBM disclosure document for an invention entitled "Information Extraction from Documents with Regular Expression Matching" was sent to the inventors' attorneys at the law firm of Ryan, Mason & Lewis, LLP, for preparation of a related patent application. The accompanying letter, dated March 14, 2001, from IBM in-house counsel Paul J. Otterstedt and the IBM disclosure document are attached hereto as Exhibit 1.

3. The IBM disclosure document was written by inventors Geoffrey G. Zweig and Mukund Padmanabhan.

4. Due diligence was performed in the preparation of a patent application from the date the letter and IBM disclosure document were received until the application was filed on July 3, 2001.

5. All statements made herein of our own knowledge are true, and all statements made on information and belief are believed to be true.

6. We understand that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001, and may jeopardize the validity of the application or any patent issuing thereon.

Date: 4/29/04 G. Zweig

Geoffrey G. Zweig

Date: _____ _____

Mukund Padmanabhan

Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

March 14, 2001

William Lewis, Esq.
Ryan, Mason & Lewis, LLP
90 Forest Avenue
Locust Valley, New York 11560

RECEIVED
WITH THANKS
RYAN, MASON & LEWIS, LLP
3.17.01

Re: IBM Docket Number: YOR920010253
Title: "INFORMATION EXTRACTION FROM DOCUMENTS WITH REGULAR EXPRESSION MATCHING"

-211

Dear Bill:

Please prepare a U.S. Patent Application for filing in the U.S. Patent and Trademark Office for the above referenced docket. This application is to be prepared in accordance with the IBM Outside Counsel Instructions in EPC format.

It is desired that this office be informed at all points involving scope of coverage and financial decisions. We wish that you handle the preparation and filing of all formal papers. Please note that the following paragraph should be used for the Power of Attorney.

POWER OF ATTORNEY: As a named inventor I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith (list name and registration number).

Manny W. Schecter (Reg. 31,722), Lauren Bruzzone (Reg. 35,082), Christopher A. Hughes (Reg. 26,914), John E. Hoel (Reg. 26,279), Joseph C. Redmond, Jr. (Reg. 18,753), Paul J. Otterstedt (Reg. 37,411), Douglas W. Cameron (Reg. 31,596), Stephen C. Kaufman (Reg. 29,551), Daniel P. Morris (Reg. 32,053), Louis J. Percello (Reg. 33,206), Marian Underweiser (Reg. 46,134), Robert M. Trepp (Reg. 25,933), Louis P. Herzberg (Reg. 41,500) Richard M. Ludwin (Reg. 33,010), Marc A. Ehrlich (Reg. 39,966), Robert P. Tassinari, Jr. (Reg. 36, 030), Derek S. Jennings (Reg. 41, 473), Gail Zarick (Reg. 43,303) and Timothy M. Farrell (Reg. 37,321).

Send Correspondence to: Outside Counsel
Direct Telephone Calls to: (name and telephone number) Outside Counsel

An Associate Power of Attorney form should be prepared and forwarded to this office for signature.

Please be advised that an additional step in our procedure is required when filing ALL original IBM Yorktown Patent Applications in the USPTO. The additional step is that a "Taiwan Oath & Assignment" form must be completed. The form must have all the required information completely filled in and must be signed and dated by all named inventors in the subject patent application. The second page of the two page form has a section entitled, "Note 1", which contains an alphabetized code reference depicting what information must be entered into each corresponding letter field (example: (a), (b), etc.). When the form is complete, and all inventor(s) signatures and dates have been obtained, please forward the original Taiwan Oath & Assignment back to my office.

Please have your illustrator prepare formal drawings for the applications. Docket number on drawings must be consistent with docket number on application. In the event these applications are filed with informal drawings, please provide us with formal drawings within six weeks after the filing date. Should you have any questions regarding the preparation of formal drawings, please contact me at (914) 945-3158.

Inventors:

Geoffrey G. Zweig	1-914-945-1204	Yorktown: 23-123
Mukund Padmanabhan	1-914-945-2929	Yorktown: 23-106D

Geoffrey G. Zweig is the Inventor Contact.

Very truly yours,

Paul J. Otterstedt

Paul J. Otterstedt
Attorney - IPLaw Dept.

PJO:jd
Enclosures
cc: Barbara Rasa
Geoffrey G. Zweig
Mukund Padmanabhan

Information Extraction from Documents with Regular Expression Matching
Geoffrey Zweig and Mukund Padmanabhan
YOR8-2000-0731

Background of the Invention

1. Field of the Invention

The present invention relates to automatic information extraction from written and spoken documents.

2. Background of the Invention

There are many situations in which it is desireable to extract key pieces of information from documents. For example, in transcribed voicemail messages, the name of the caller and any return numbers that were left are crucial for summarizing the call. Or when resumes are submitted to a company along with a cover letter, it is desireable to extract the job objective and salary requirements of the applicant, in order to determine if a suitable match exists. This invention comprises a simple, efficient, and effective way of extracting such information.

In the past, several techniques have been developed to solve the somewhat simpler problem of "named entity extraction." In this task, the goal is to identify all occurrences of certain classes of words in a document. For example, all the person-names, city-names, dates, and times might be identified. One way of identifying such entities is to train a statistical classification system to tag each word in the document as either a "person-name," "city-name," "date," "time," or "other" word. Examples of this sort of approach can be found in, e.g. Ratnaprkhi ("A Maximum Entropy Part of Speech Tagger," In Eric Brill and Kenneth Church, editors, *Conference on Empirical Methods in Natural Language Processing, University of Pennsylvania*, 1996) and Brill ("Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging," in *Computational Linguistics*, Dec. 1995). While the problem of extracting key pieces of information can also be viewed as a tagging problem - in which each word is tagged as either "key information" or "irrelevant," the present invention takes a more direct approach and exploits the readily-identifiable structure of language to explicitly identify the portions of text that are important.

In common linguistic usage, people usually convey information in highly stereotyped ways. So, for example, in a phone call, the caller is likely to identify himself in one of just a few ways, e.g. "Hi <recipient-name> it's <caller-name>" or "<recipient-name> <caller-name> here." Or in a cover letter, a job applicant is likely to express his interest with a phrase like "I am looking for a job in <field>."

The basic idea of this invention is to enumerate all the stereotypical phrases that people use to convey important information, and where exactly in these phrases the inormation is to be found. This approach has several advantages over the previously mentioned tagging approach. First, the phrases are readily expressible as regular expressions, and this allows them to be stored in one of several common programming langauages, e.g. As a "flex" or "perl" program. Both these langauges have built-in support for regular expressions. Secondly, the use of regular-expression technology enables extremely fast pattern matching and information extraction with highly optimized standard programs. Thirdly, the phrases can be identified without the expensive and time consuming step of gathering and annotating a large "training" database. Finally, if desired, the technique can be combined with statistical based procedures, e.g. By using

the phrases as features in a statistical system, or by using a statistical procedure to automatically collect a large pool of candidate phrases which can then be selected for inclusion by a human.

Summary of the Invention

This invention consists of using regular expressions to identify the information-bearing portions of a document, and the explicit identity of the information found therein. Regular expressions (Aho Hopcroft and Ullman) consist of symbolic strings, typically word sequences, that can be constructed by the basic operations of "and" and "or" acting on a atomic alphabet of symbols. So, for example, the pattern "*The dog*" will match only when the symbol the "the" is followed by the symbol "dog." This represents the operation of concatenation or "and." The pattern "*The (dog | cat)*" illustrates the "or" operation: it will match either the string "The dog" or "The cat." The operation of "classing" is also possible, so, for example, the "pet" class may be defined as either "dog" or "cat:"
<animal> ::= (dog | cat)
And then subsequent regular expressions can make use of this class, for example, the pattern: "*The <pet>.*" A wide variety of extensions to regular expressions exist, for example, restrictions on where in a document a pattern is found, or restrictions on the length of repeated sequences in a regular expression, and in this patent, the term "regular expression" is taken to mean any sort of pattern that is matchable in the *flex* and *perl* programming languages. Similarly, the term "matching" refers to the processes by which these programs match their input.

In this invention, when an information-bearing regular expression is matched, an action is taken to display the information in an appropriate way. There are two main types of regular expressions that are useful: trigger phrases and trigger suffixes. A trigger prefix is a characteristic phrase that typically precedes a piece of information, for example in a voicemail message, "give me a call back at" is a trigger-phrase that precedes a phone number. A trigger suffix is a phrase that typically follows a key piece of information, even when a trigger prefix is not present. For example the phrase "talk to you later bye" is often a post-facto signal that a phone number has been provided, when the words immediately preceding it are a sequence of numbers. In this invention, trigger prefixes and suffixes may be used either separately, or combined together into larger composite patterns. The operation of word classing is also inherent in this invention. For example, a set of personal names such as those found in a phone book may be identified a a "person-name" class. Subsequently, patterns of the form *"Hi it's <person-name>+"* can be matched, and the words matching the "person-name" tokens can be displayed as the caller. It will be understood by someone skilled in the art that there are numerous ways of combining trigger prefixes and suffixes, word classes, and other pattern-matching operations such as the the use of multiple states each with its own set of matchable patterns (common in *flex*) in order to identify and extract information.

In addition to being used as a stand-alone method for information extraction, regular expressions can be in combination with statistical techniques. For example, the existence of a pattern such as "*hi <person-name> it's <person-name>*" can be used as a feature in a maximum-entropy word-tagging system.

Brief Description of the Drawings

Figure 1 illustrates the sequence of operations performed in a preferred embodiment of this invention. Figure 2 presents a *flex* program suitable for information extraction from voicemail messages.

Detailed Description of the Preferred Embodiment

Figure 1 illustrates the information processing performed in a preferred embodiment of this invention. In an initial step, the input is normalized. This consists of standardizing any characteristics of the input that are irrelevant to, or a hindrance to, information extraction. An example of normalization, for text input, is to change all the letters to lower-case and to replace multiple blank spaces between words by a single blank spaces. This makes subsequent pattern matching easier. In the second step, one or more word (or in general symbol) lists are used to identify and mark occurrences of words that belong to certain classes. For example, all proper names might be marked by prefixing them with an "!" Mark.

In the next step, the normalized and class-annotated text is read by a regular-expression matching program, and the information-bearing portions of the text are analyzed. In a preferred embodiment,
The regular expressions are specified in and matched by a specialized regular expression matching program such as *flex*. In this process, it is understood that all the mechanisms available in such a program may be used. These include the use of special start states, backtracking, context-sensitive matching, and embedded special-purpose "C" code.

Figure 2 presents an illustratory *flex* program for extracting caller names and numbers from voicemail messages. This program is meant to be purely illustratory, and one skilled in the art will be able to create alternative embodiments. Similarly, the same basic regular-expression matching ideas can be used to identify information in other sorts of documents.

It will also be understood that this invention applies to many input forms besides typed text. For example, it applies to information extraction from any data stream that can be symbolically represented, for example: DNA sequences, RNA sequences, amino-acid sequences, and audio and video sequences that are pre-processed into a discrete symbolic form.

What is Claimed is:

1. A process by which a set of regular expressions are used to extract information from documents. In this context, "document" refers to any symbolic sequence and encompasses, for example, written text and DNA sequences.

2. A process based on 1) in which specified actions are taken when regular expressions are matched.

3. A process based on 2) in which the regular expressions include trigger prefixes or trigger suffixes.

4. A process beased on 1) in which word classes are used.

5. A process based on 1) in which the input is normalized before the regular expressions are matched.

6. A process based on 1) in which information is extracted from voicemail messages.

7. A process based on 1) in which a flex, lex, perl, or other standardly available program is used to match the regular expressions.

Figure 1: Information Extraction Steps

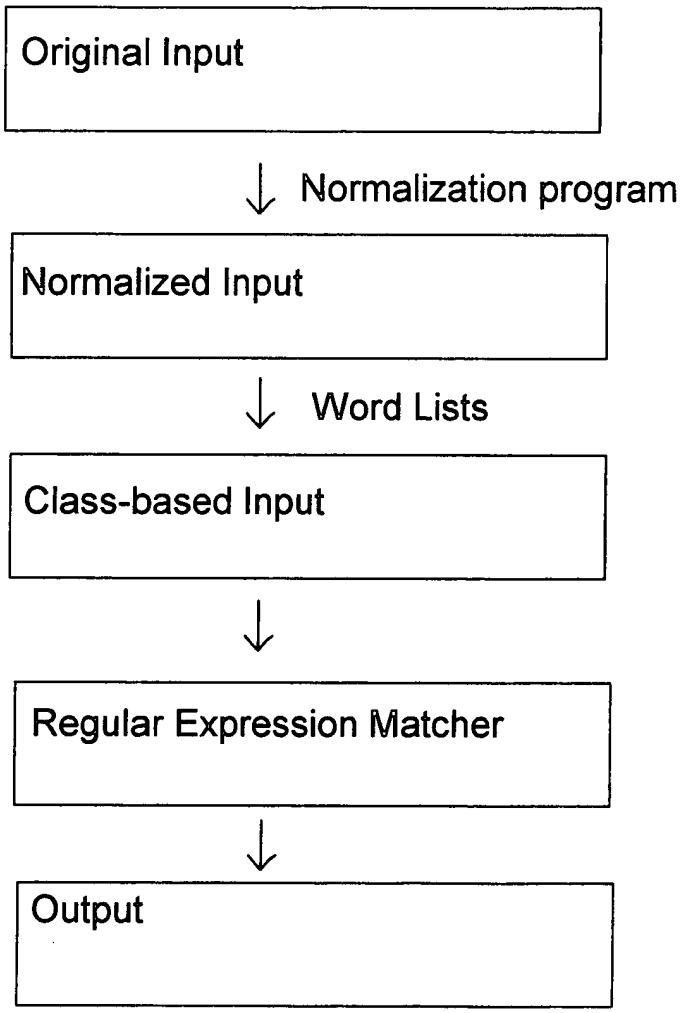

Figure 2: A Sample *flex* Program for Normalizing Text Input

```
%{

#include <iostream.h>
#include <ctype.h>

%}

%option noyywrap
%option never-interactive

ws [ \t]+

%%

-                   {cout << " ";}
[!\.@,"()?]             {;}
\<[^\>]*\>{ws}*         {;}
{ws}                {cout << " ";}
.                   {cout << char(tolower(*yytext));}

%%

main()
{
        yylex();
}
```

Figure 3: A C++ Program for Identifying Names in a Text Stream

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <set>
#include <string>

main()
{
        set<string> name;
        ifstream namesin("../data/person_names");  // a text file listing names
        if (!namesin)
        {
                cout << "Unable to find ../data/person_names\n";
                exit(1);
        }

        while (!namesin.eof())
        {
                string s;
                namesin >> s >> ws;
                name.insert(s);
        }
        namesin.close();

        while (!cin.eof())
        {
                string s;
                cin >> s >> ws;
                if (name.count(s))
                        cout << "!" << s << " ";
                else
                        cout << s << " ";
        }
}
```

Figure 4: A *flex* Program for Identifying Caller-Names and Phone-Numbers in Voicemail Messages

```
%{

#include <iostream.h>
#include <string.h>

const char *number_type;
const char *tl = "tieline ";
const char *extension = "extension ";
const char *pager = "pager ";
const char *fax = "fax number";
const char *none = "";

char number_buff[10000];

%}

%option noyywrap
%option never-interactive

 /* many patterns like IN-GENERIC_NAME-AT-DIGIT+ unaccounted for */

spelled_digit
((zero|oh|one|two|three|four|five|six|seven|eight|nine|hundred|thousand|ten|eleven|twelve|thirteen|fou
rteen|fifteen|sixteen|seventeen|eighteen|nineteen|twenty|thirty|forty|fifty|sixty|seventy|eighty|ninety|
hundred)[ ]?)
numeric_digit ((1|2|3|4|5|6|7|8|9|10)[ ]?)
digit ([ ]?{spelled_digit}|[ ]?{numeric_digit})
phone_number ((country[ ]code[ ]{digit}{2,})?(toll[ ]free)?(area[ ]|area[ ]code[
])?{digit}{2,}((or[ ]|that's[ ]|sorry[ ]){digit}{2,})?(option[ ]{digit}+)?(extension[ ]{digit}{2,})?)
tie_line ([ ]?(tie[ ]line|tieline)([ ]number)?)
name ![a-z']+

%x PHONE_NUMBER_STATE
%x NAME_STATE
%x EXTENDED_NAME_STATE
%x TERMINATION_STATE

%%

(1|2|3|4|5|6|7|8|9|0)+": " {cout << yytext << endl; /* get header */ }

{digit}({digit}|"extension "){3,} {strcpy(number_buff, yytext); BEGIN
TERMINATION_STATE;}
```

```
<TERMINATION_STATE>"thank you " |
<TERMINATION_STATE>"thanks " |
<TERMINATION_STATE>"take care " |
<TERMINATION_STATE>"bye " |
<TERMINATION_STATE>"talk to you " |
<TERMINATION_STATE>"okay bye " |
<TERMINATION_STATE>"have a "(good|great)" day "  {cout << "Number: " << number_buff
<< endl; BEGIN INITIAL;}
<TERMINATION_STATE><<EOF>> {cout << "Number: " << number_buff << endl; BEGIN
INITIAL;}
<TERMINATION_STATE>. {unput(*yytext); BEGIN INITIAL;}

"on extension " |
"at extension " |
"extension " |
"extension is "   {number_type = extension; BEGIN PHONE_NUMBER_STATE;}

"my "{tie_line}" is " |
"my "{tie_line}" here is " |
{tie_line}" is " |
{tie_line}" "     {number_type = tl; BEGIN PHONE_NUMBER_STATE;}

"again is " |
"again " |
"back " |
"call me " |
"call me at " |
"call me back at " |
"call me "[a-zA-Z]+" at " |
"call me back on " |
"call me on " |
"call back at " |
"call back on " |
"give me a buzz " |
"give me a buzz at " |
"give me a holler " |
"give me a holler at " |
"give me a ring " |
"give me a call " |
"give me a ring at " |
"give me a call at " |
"give me a call at home " |
"give me a call at work " |
"give me a call at your convenience " |
"give me a call back at " |
```

```
"give me a call back on " |
"give me a call back with that information at " |
"give me a call if you get a chance " |
"give me a call on " |
"give me a call please " |
"give me a call please at " |
"give me a call when you get a chance " |
"give me a call when you can " |
"give me a call when you can please " |
"give us a call " |
"give us a call at " |
"here " |
"i can be reached at " |
"i am at " |
"i'm at " |
"i'll be at " |
"i'm on " |
"leave me a message " |
"leave me a message at " |
"my number is " |
"my number here is " |
"my number here in "(!?[a-z]+[ ])"is " |
"my number's " |
"my phone number's " |
"number again is " |
"number at home is " |
"number at work is " |
"number here is " |
"number is " |
"number's " |
"outside line " |
"outside " |
"please call " |
"phone " |
"please return our call at " |
"reach me at " |
"returning your call " |
"talk to you later " |
"try me at " |
"try me at this number " |
"you could reach us at " |
"when you have a chance " |
"when you get a chance " {number_type=none; BEGIN PHONE_NUMBER_STATE;}

"fax number is " |
"my fax number's" {number_type=fax; BEGIN PHONE_NUMBER_STATE;}
```

```
"my pager " |
"my pager is " |
"give me a page " |
"give me a page at " |
"page me at " |
"page me " |
"page number "          {number_type=pager; BEGIN PHONE_NUMBER_STATE;}

<PHONE_NUMBER_STATE>{phone_number} {cout << "Number: " << number_type <<
yytext << endl; BEGIN INITIAL;}
<PHONE_NUMBER_STATE>. {unput(*yytext); BEGIN INITIAL;}

"it's "{phone_number} |
"on "{phone_number} {char *c=strchr(yytext, ' '); c++; cout << "Number: " << c << endl;}

"it's me "   |
"it's just me "      {cout << "Name: me" << endl;}
"this is your wife " {cout << "Name: wife" << endl;}
"this is receiving " {cout << "Name: receiving" << endl;}
"this is the mail room " {cout << "Name: the mail room" << endl;}

^({name}[ ]){2,3}  {cout << "Name: " << (strchr(yytext, ' ')+1) << endl;}
("hi "|"hey "|"hello "){name}" "{name}" "({name}" ")?    {char *c=strchr(yytext, ' ');
c=strchr(c+1, ' '); cout << "Name: " << (c+1) << endl;}

"good morning "{name}" " |
"yeah "{name}" " |
({name}[ ])?"this is " |
{name}" it's " |
{name}" hi " |
"hi i'm " |
"my name is " |
"my name's " |
"the name is " |
"the name's " |
"hey "({name}[ ])?"this "("is ")? |
"hi "({name}[ ])?"this "("is ")? |
({name}[ ])?"hi it's " |
({name}[ ])?"hey it's " |
({name}[ ])?"how you doing it's " |
{name}" this "              {BEGIN NAME_STATE;}

<NAME_STATE>({name}[ ])+ {cout << "Name: " << yytext << endl; number_type=none;
BEGIN PHONE_NUMBER_STATE;}
```

```
<NAME_STATE>("me "|"mom "|"your sis"|"your sister"|"your wife"|"your brother"|"your
father"|"your husband") {cout << "Name: " << yytext << endl; BEGIN INITIAL;}

<NAME_STATE>({name}[ ])+"from " |
<NAME_STATE>({name}[ ])+"calling from " |
<NAME_STATE>({name}[ ])+"i'm calling from " |
<NAME_STATE>({name}[ ])+"with " |
<NAME_STATE>({name}[ ])+"i'm with " |
<NAME_STATE>({name}[ ])+"at " |
<NAME_STATE>({name}[ ])+"over at " |
<NAME_STATE>({name}[ ])+"down at " |
<NAME_STATE>({name}[ ])+"over in " |
<NAME_STATE>({name}[ ])+"up in " |
<NAME_STATE>({name}[ ])+"down in " |
<NAME_STATE>({name}[ ])+"up on " |
<NAME_STATE>({name}[ ])+"down on " |
<NAME_STATE>({name}[ ])+"in " {cout << "Name: " << yytext; BEGIN
EXTENDED_NAME_STATE;}
<NAME_STATE>. {unput(*yytext); BEGIN INITIAL;}

<EXTENDED_NAME_STATE>" i'm " |
<EXTENDED_NAME_STATE>" i " |
<EXTENDED_NAME_STATE>" i've " |
<EXTENDED_NAME_STATE>" i'd " |
<EXTENDED_NAME_STATE>" my " |
<EXTENDED_NAME_STATE>" it's " |
<EXTENDED_NAME_STATE>" it is " |
<EXTENDED_NAME_STATE>" we " |
<EXTENDED_NAME_STATE>" when " |
<EXTENDED_NAME_STATE>" you " |
<EXTENDED_NAME_STATE>" calling " |
<EXTENDED_NAME_STATE>" now " |
<EXTENDED_NAME_STATE>" a " |
<EXTENDED_NAME_STATE>" if " |
<EXTENDED_NAME_STATE>" give " |
<EXTENDED_NAME_STATE>" their " |
<EXTENDED_NAME_STATE>" and " |
<EXTENDED_NAME_STATE>" notifying " |
<EXTENDED_NAME_STATE>" returning " |
<EXTENDED_NAME_STATE>" sorry " |
<EXTENDED_NAME_STATE>" can " |
<EXTENDED_NAME_STATE>" got " |
<EXTENDED_NAME_STATE>" earlier " |
<EXTENDED_NAME_STATE>" your " |
<EXTENDED_NAME_STATE>" just " |
<EXTENDED_NAME_STATE>" that " |
```

```
<EXTENDED_NAME_STATE>" could " |
<EXTENDED_NAME_STATE>" would " |
<EXTENDED_NAME_STATE>" before " |
<EXTENDED_NAME_STATE>" thanks " |
<EXTENDED_NAME_STATE>" thank " |
<EXTENDED_NAME_STATE>" wanted " |
<EXTENDED_NAME_STATE>" following " |
<EXTENDED_NAME_STATE>" regarding " |
<EXTENDED_NAME_STATE>" may " |
<EXTENDED_NAME_STATE>" let " |
<EXTENDED_NAME_STATE>" he " |
<EXTENDED_NAME_STATE>" about " |
<EXTENDED_NAME_STATE>" again " |
<EXTENDED_NAME_STATE>" wanting " |
<EXTENDED_NAME_STATE>" appreciate " |
<EXTENDED_NAME_STATE>" was " |
<EXTENDED_NAME_STATE>" we're " |
<EXTENDED_NAME_STATE>" or " |
<EXTENDED_NAME_STATE>" you've " |
<EXTENDED_NAME_STATE>" right " |
<EXTENDED_NAME_STATE>" first " |
<EXTENDED_NAME_STATE>[ ]?"tieline " |
<EXTENDED_NAME_STATE>[ ]?"extension " |
<EXTENDED_NAME_STATE>" "{digit} |
<EXTENDED_NAME_STATE>" please " {cout << endl; char *q=yytext+strlen(yytext); while
(q>yytext) unput(*--q); number_type=none; BEGIN PHONE_NUMBER_STATE;}
<EXTENDED_NAME_STATE>. {ECHO;}

. {;}

%%

main()
{
        yylex();
        cout << endl;
}
```